Введение в параллельные вычисления. Технология программирования MPI

(день четвертый)

Антонов Александр Сергеевич, к. ф.-м.н., н.с. лаборатории Параллельных информационных технологий НИВЦ МГУ

MPI

```
MPI_SEND_INIT(BUF, COUNT,
DATATYPE, DEST, MSGTAG, COMM,
REQUEST, IERR)
<type> BUF(*)
INTEGER COUNT, DATATYPE, DEST,
MSGTAG, COMM, REQUEST, IERR
```

Формирование отложенного запроса на посылку сообщения. Сама операция пересылки не начинается!

MPI

Модификации функции **MPI_SEND_INIT:**

- **MPI_BSEND_INIT** — формирование запроса на передачу сообщения с буферизацией.
- **MPI_SSEND_INIT** — формирование запроса на передачу сообщения с синхронизацией.
- **MPI_RSEND_INIT** — формирование запроса на передачу сообщения по готовности.

MPI

```
MPI_RECV_INIT(BUF, COUNT,
DATATYPE, SOURCE, MSGTAG, COMM,
REQUEST, IERR)
<type> BUF(*)
INTEGER COUNT, DATATYPE, SOURCE,
MSGTAG, COMM, REQUEST, IERR
```

Формирование отложенного запроса на прием сообщения. Сама операция приема не начинается!

MPI

```
MPI_START(REQUEST, IERR)
INTEGER REQUEST, IERR
```

Инициализация отложенного запроса на выполнение операции обмена, соответствующей значению параметра **REQUEST**. Операция запускается как неблокирующая.

MPI

```
MPI_STARTALL(COUNT, REQUESTS, IERR)
INTEGER COUNT, REQUESTS, IERR
```

Инициализация **COUNT** отложенных запросов на выполнение операций обмена, соответствующих значениям первых **COUNT** элементов массива **REQUESTS.** Операции запускаются как неблокирующие.

MPI

Сообщение, отправленное при помощи отложенного запроса, может быть принято любой из процедур **MPI_RECV** и **MPI_IRECV**, и наоборот.

По завершении отложенного запроса значение параметра **REQUEST (REQUESTS)** сохраняется и может использоваться в дальнейшем!

MPI

MPI_REQUEST_FREE(REQUEST, IERR)

INTEGER REQUEST, IERR

Удаляет структуры данных, связанные с параметром **REQUEST. REQUEST** устанавливается в значение **MPI_REQUEST_NULL**. Если операция, связанная с этим запросом, уже выполняется, то она завершится.

MPI

```
prev = rank - 1
next = rank + 1
if (rank .eq. 0) prev = numtasks - 1
if (rank .eq. numtasks - 1) next = 0
call MPI_RECV_INIT(rbuf(1), 1,
  &      MPI_REAL, prev, tag1,
  &      MPI_COMM_WORLD, reqs(1), ierr)
call MPI_RECV_INIT(rbuf(2), 1,
  &      MPI_REAL, next, tag2,
  &      MPI_COMM_WORLD, reqs(2), ierr)
call MPI_SEND_INIT(sbuf(1), 1,
  &      MPI_REAL, prev, tag2,
  &      MPI_COMM_WORLD, reqs(3), ierr)
```

MPI

```
call MPI_SEND_INIT(sbuf(2), 1,
  &      MPI_REAL, next, tag1,
  &      MPI_COMM_WORLD, reqs(4), ierr)
   do i=...
      sbuf(1)=...
      sbuf(2)=...
      call MPI_STARTALL(4, reqs, ierr)
      ...
    call MPI_WAITALL(4, reqs, stats,
  &          ierr);
      ...
   end do
```

MPI

Тупиковые ситуации (deadlock):

процесс 0:	процесс 1:
`RECV(1)`	`RECV(0)`
`SEND(1)`	`SEND(0)`

процесс 0:	процесс 1:
`SEND(1)`	`SEND(0)`
`RECV(1)`	`RECV(0)`

MPI

Разрешение тупиковых ситуаций:

1.

процесс 0:	процесс 1:
SEND(1)	**RECV(0)**
RECV(1)	**SEND(0)**

2. Использование неблокирующих операций

3. Использование функции **MPI_SENDRECV**

MPI

```
MPI_SENDRECV(SBUF, SCOUNT,
STYPE, DEST, STAG, RBUF, RCOUNT,
RTYPE, SOURCE, RTAG, COMM,
STATUS, IERR)
<type> SBUF(*), RBUF(*)
INTEGER SCOUNT, STYPE, DEST,
STAG, RCOUNT, RTYPE, SOURCE,
RTAG, COMM,
STATUS(MPI_STATUS_SIZE), IERR
```

MPI

Совмещенные прием и передача сообщений с блокировкой. Буферы передачи и приема не должны пересекаться. Тупиковой ситуации не возникает!

Сообщение, отправленное операцией **`MPI_SENDRECV`**, может быть принято обычным образом, и операция **`MPI_SENDRECV`** может принять сообщение, отправленное обычной операцией.

MPI

```
MPI_SENDRECV_REPLACE(BUF, COUNT,
DATATYPE, DEST, STAG, SOURCE,
RTAG, COMM, STATUS, IERR)
<type> BUF(*)
INTEGER COUNT, DATATYPE, DEST,
STAG, SOURCE, RTAG, COMM,
STATUS(MPI_STATUS_SIZE), IERR
```

Совмещенные прием и передача сообщений с блокировкой через общий буфер **BUF**.

MPI

```
prev = rank - 1
next = rank + 1
if (rank .eq. 0) prev = numtasks - 1
if (rank .eq. numtasks - 1) next = 0
call MPI_SENDRECV(
  &      sbuf(1), 1, MPI_REAL, prev, tag2,
  &      rbuf(1), 1, MPI_REAL, next, tag2,
  &      MPI_COMM_WORLD, status1, ierr)
call MPI_SENDRECV(
  &      sbuf(2), 1, MPI_REAL, next, tag1,
  &      rbuf(2), 1, MPI_REAL, prev, tag1,
  &      MPI_COMM_WORLD, status1, ierr)
```

MPI

MPI_BARRIER(COMM, IERR)

INTEGER COMM, IERR

Работа процессов блокируется до тех пор, пока все оставшиеся процессы коммуникатора **COMM** не выполнят эту процедуру. Все процессы должны вызвать **MPI_BARRIER**, хотя реально исполненные вызовы различными процессами коммуникатора могут быть расположены в разных местах программы.

MPI

Специальное значение **MPI_PROC_NULL** для несуществующего процесса. Операции с таким процессом завершаются немедленно с кодом завершения **MPI_SUCCESS**.

```
    next=rank+1
    if(rank .EQ. size-1) next=MPI_PROC_NULL
call MPI_SEND (buf, 1, MPI_REAL, next,
  &      tag, MPI_COMM_WORLD, ierr)
```

MPI

```
MPI_GET_PROCESSOR_NAME(NAME,
LEN, IERR)
CHARACTER*(*) NAME
INTEGER LEN, IERR
```

Функция возвращает в строке **NAME** имя узла, на котором запущен вызвавший процесс. В переменной **LEN** возвращается количество символов в имени, не превышающее константы **MPI_MAX_PROCESSOR_NAME**.